ANL-85-19

COMPARISON OF THE CRAY X-MP-4, FUJITSU VP-200, AND HITACHI S-810/20: AN ARGONNE PERSPECTIVE

by

Jack J. Dongarra and Alan Hinds

ARGONNE NATIONAL LABORATORY, ARGONNE, ILLINOIS

Operated by THE UNIVERSITY OF CHICAGO
for the U. S. DEPARTMENT OF ENERGY
under Contract W-31-109-Eng-38

Argonne National Laboratory, with facilities in the states of Illinois and Idaho, is owned by the United States government, and operated by The University of Chicago under the provisions of a contract with the Department of Energy.

Printed in the United States of America
Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

Distribution Category:
Mathematics and Computers
General (UC-32)

ANL-85-19

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

Comparison of the CRAY X-MP-4, Fujitsu VP-200, and Hitachi S-810/20: An Argonne Perspective

Jack J. Dongarra
Mathematics and Computer Science Division

and

Alan Hinds
Computing Services

October 1985

Table of Contents

List of Tables

List of Figures

Comparison of the CRAY X-MP-4, Fujitsu VP-200, and Hitachi S-810/20: An Argonne Perspective*

Jack J. Dongarra
Mathematics and Computer Science Division

Alan Hinds
Computing Services

Abstract

A set of programs, gathered from major Argonne computer users, was run on the current generation of supercomputers: the CRAY X-MP-4, Fujitsu VP-200, and Hitachi S-810/20. The results show that a single processor of a CRAY X-MP-4 is a consistently strong performer over a wide range of problems. The Fujitsu and Hitachi excel on highly vectorized programs and offer an attractive opportunity to sites with IBM-compatible computers.

1. Introduction

Last year we ran a set of programs, gathered from major Argonne computer users, on the current generation of supercomputers: the CRAY X-MP-4 at CRAY Research in Mendota Heights, Minnesota; the Fujitsu VP-200 at the Fujitsu plant in Numazu, Japan; and the Hitachi S-810/20 at the Hitachi Ltd. Kanagawa Works in Kanagawa, Japan.

2. Architectures

The CRAY X-MP, Fujitsu VP, and Hitachi S/810 computers are all high-performance vector processors that use pipeline techniques in both scalar and vector operations and provide parallelism among independent functional units. All three machines use a register-to-register format for instruction execution and are architecturally similar at a high level. Each machine has three vector load/store techniques — contiguous element, constant stride, and indirect address (index vector) modes. All three are optimized for 64-bit floating-point arithmetic operations. There are, however, a number of major differences that should be noted. These are discussed below and summarized in Table 1 at the end of this section.

2.1 CRAY X-MP

The CRAY X-MP-4 is the largest of the family of CRAY X-MP computer models, which range in size from one to four processors and from one million to sixteen million words of central memory. The CRAY X-MP/48 computer consists of four identical pipelined processors, each with fully

*Work supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

segmented scalar and vector functional units with a 9.5-nanosecond clock cycle. All four processors share in common an 8-million word, high-speed (38-nanosecond cycle time) bipolar central memory, a common I/O subsystem, and an optional integrated solid-state storage device (SSD). Each processor contains a complete set of registers and functional units, and each processor can access all of the common memory, all of the I/O devices, and the single (optional) SSD. The four CPUs can process four separate and independent jobs, or they can be organized to work concurrently on a single job. The architecture of the CRAY X-MP/48 is depicted in Figure 1.

This document will focus on the performance of only a single processor of the CRAY X-MP-4, as none of our benchmark programs were organized to take advantage of multiple processors. Thus, in the tables and text that follow, all data on the capacity and the performance of the CRAY X-MP-4 apply to a single processor, except for data on the size of memory and the configuration and performance of I/O devices and the SSD.

The CRAY X-MP-4 has extremely high floating-point performance for both scalar and vector applications and both short and long vector lengths. Each CRAY X-MP-4 processor has a maximum theoretical floating-point result rate of 210 MFLOPS (millions of floating point operations per second) for overlapped vector multiply and add instructions. With the optional solid-state storage device installed, the CRAY X-MP-4 has an input/output bandwidth of over 2.4 billion bytes per second, the largest in this study; without the SSD, the I/O bandwidth is 424 million bytes per second, of which 68 million bytes per second is attainable by disk I/O. The CRAY permits a maximum of four disk devices on each of eight disk control units, the smallest disk subsystem in this study.

The cooling system for the CRAY X-MP-4 is refrigerated liquid freon.

The CRAY X-MP-4 operates with the CRAY Operating System (COS), a batch operating system designed to attach by a high-speed channel or hyperchannel interface with a large variety of self-contained, general-purpose front-end computers. All computing tasks other than batch compiling, linking, and executing of application programs must be performed on the front-end computer. Alternatively, the CRAY X-MP-4 can operate under CTSS (CRAY Time-Sharing System, available from Lawrence Livermore National Laboratory), a full-featured interactive system with background batch computing. The primary programming languages for the CRAY X-MP are Fortran 77 and CAL (CRAY Assembly Language); the Pascal and C programming languages are also available.

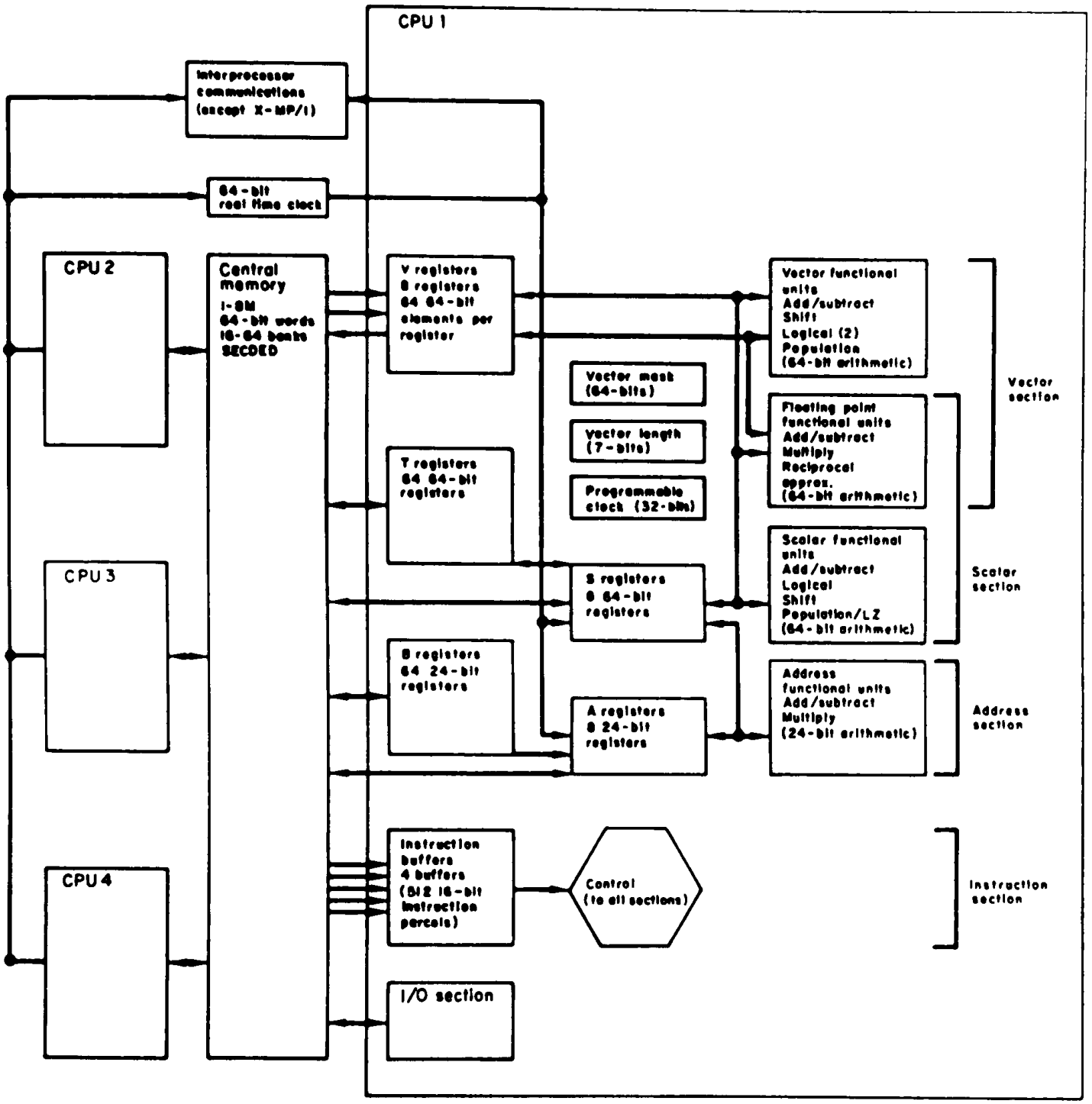

Figure 1
CRAY X-MP/48 Architecture

2.2 Fujitsu VP-200 (Amdahl 1200)

The Fujitsu VP-200 is midway in performance in a family of four Fujitsu VP computers, whose performance levels range to over a billion floating-point operations per second. In North America, the Fujitsu VP-200 is marketed and maintained by the Amdahl Corporation as the Amdahl 1200 Vector Processor. Although we benchmarked the VP-200 in Japan, the comparisons in this document will emphasize the configurations of the VP-200 offered by Amdahl in the United States.

The Fujitsu VP-200 is a high-speed, single-processor computer, with up to 32 million words of fast (60-nanosecond cycle time) static MOS central memory. The VP-200 has separate scalar (15-nanosecond clock cycle) and vector (7.5-nanosecond clock cycle) execution units, which can execute instructions concurrently. A unique characteristic of the VP-200 vector unit is its large (8192-word) vector register set, which can be dynamically configured into different numbers and lengths of vector registers. The VP-200 architecture is depicted in Figure 2.

The VP-200 has a maximum theoretical floating point result rate of 533 MFLOPS for overlapped vector multiply and add instructions.

The VP-200 system is cooled entirely by forced air.

The Fujitsu VP-200 scalar instruction set and data formats are fully compatible with the IBM 370 instruction set and data formats; the VP-200 can execute load modules and share load libraries and datasets that have been prepared on IBM-compatible computers. The Fujitsu VP-200 uses IBM-compatible I/O channels and can attach all IBM-compatible disk and tape devices and share these devices with other IBM-compatible mainframe computers. Fujitsu does not offer an integrated solid-state storage device for the VP computer series, but any such device that attaches to an IBM channel and emulates an IBM disk device can be attached to the VP-200. The total I/O bandwidth of the VP-200 is 96 million bytes per second, the smallest in this study. The VP-100 can attach over one thousand disks; up to 93 million bytes per second can be used for disk I/O.

The Fujitsu VP-200 operates with the FACOM VP control program (also called VSP — Vector Processor System Program — by Amdahl), a batch operating system designed to interface with an IBM-compatible front-end computer via a channel-to-channel (CTC) adaptor in a tightly coupled or loosely coupled network. The front-end computer operating system may be Fujitsu's OS-IV (available only in Japan) or IBM's MVS, MVS/XA, or VM/CMS. To optimize use of the VP vector hardware, Fujitsu encourages VP users to perform all computing tasks, other than executing their Fortran application programs, on the front-end computer.

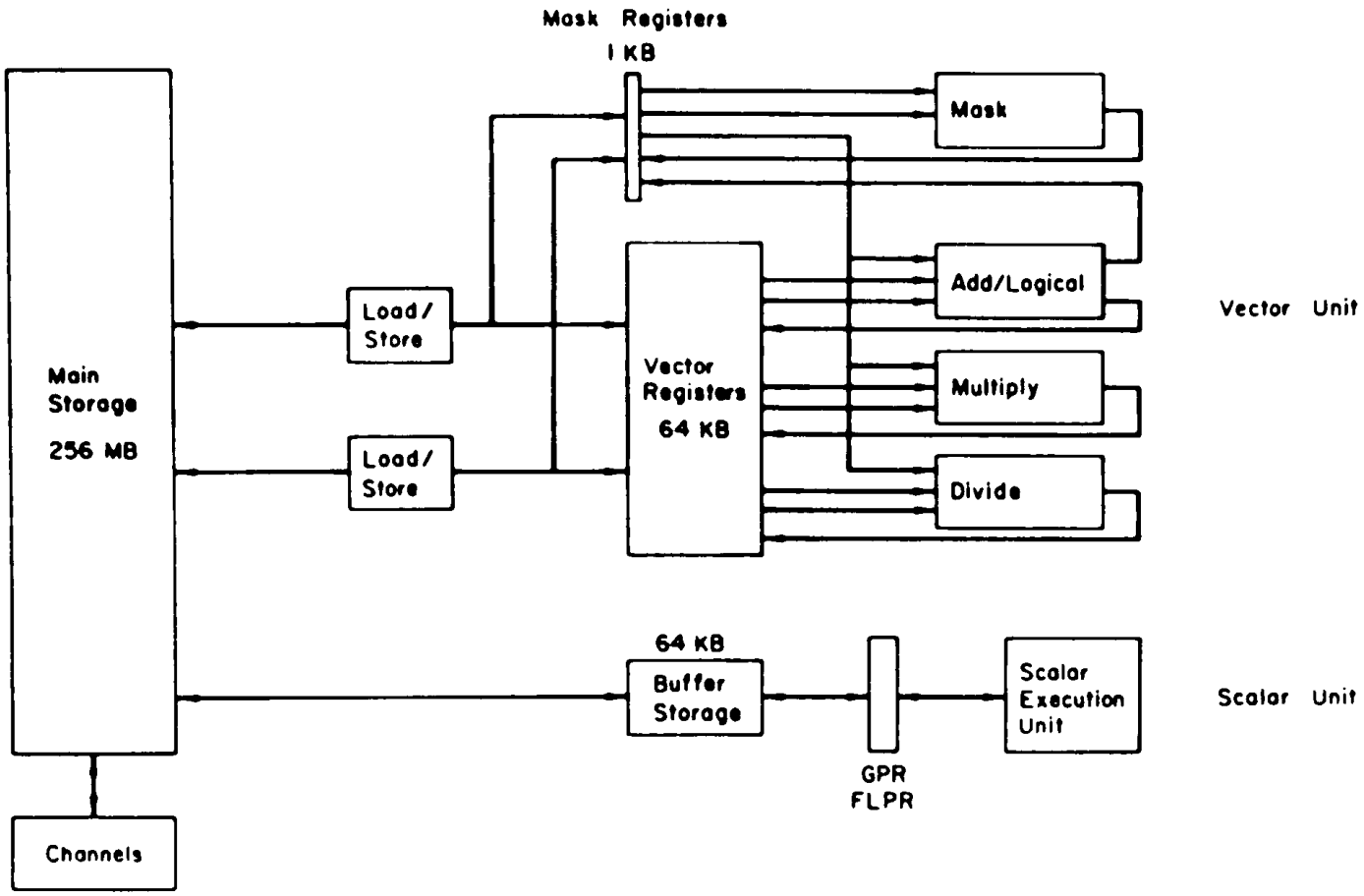

Figure 2
Fujitsu VP-200 Architecture

Of the three machines in this study, Fujitsu (Amdahl) provides the most powerful set of optimizing and debugging tools, all of which run on the front-end computer system. The only programming language that takes advantage of the Fujitsu VP vector capability is Fujitsu Fortran 77/VP, although object code produced by any other compiler or assembler available for IBM scalar mainframe computers will execute correctly on the VP in scalar mode.

2.3 Hitachi S-810/20

The Hitachi S-810/20 computer is the more powerful of two Hitachi S-810 computers, which currently are sold only in Japan. Very little is published in English about the Hitachi S-810 computers; consequently, some data in the tables and comparisons are inferred and may be inaccurate.

The Hitachi S-810/20 is a high-speed, single-processor computer, with up to 32 million words of fast (70-nanosecond bank cycle time) static MOS central memory and up to 128 million words of extended storage. The computer has separate scalar (28-nanosecond clock cycle) and vector (14-nanosecond clock cycle) execution units, which can execute instructions concurrently. The scalar execution unit is distinguished by its large (32 thousand words) cache memory. The S-810/20 vector unit has 8192 words of vector registers, and the largest number of vector functional units and the most comprehensive vector macro instruction set of the three machines in this study. The Hitachi S-810 family has the unique ability to process vectors that are longer than their vector registers, entirely under hardware control. The architecture is shown in Figure 3.

The Hitachi S-810/20 has a maximum theoretical floating point result rate of 840 MFLOPS for overlapped vector multiply and add instructions (four multiply and eight add results per cycle).

The S-810/20 computer is cooled by forced air across a closed, circulating water radiator.

Like the Fujitsu VP, the Hitachi S-810/20 scalar instruction set and data formats are fully compatible with the IBM 370 instruction set and data formats; the S-810/20 can execute load modules and share load libraries and datasets that have been prepared on IBM-compatible computers. The Hitachi S-810/20 uses IBM-compatible I/O channels and can attach all IBM-compatible disk and tape devices and share these devices with other IBM-compatible mainframe computers. Hitachi's optional, extended storage offers extremely high performance I/O. With the extended storage installed, the Hitachi S-810/20 has an I/O bandwith of 1.1 billion bytes per second; without extended storage the I/O bandwidth is 96 million bytes per second. Up to 93 million bytes per second can be used for disk I/O. The Hitachi can attach over one thousand disk devices.

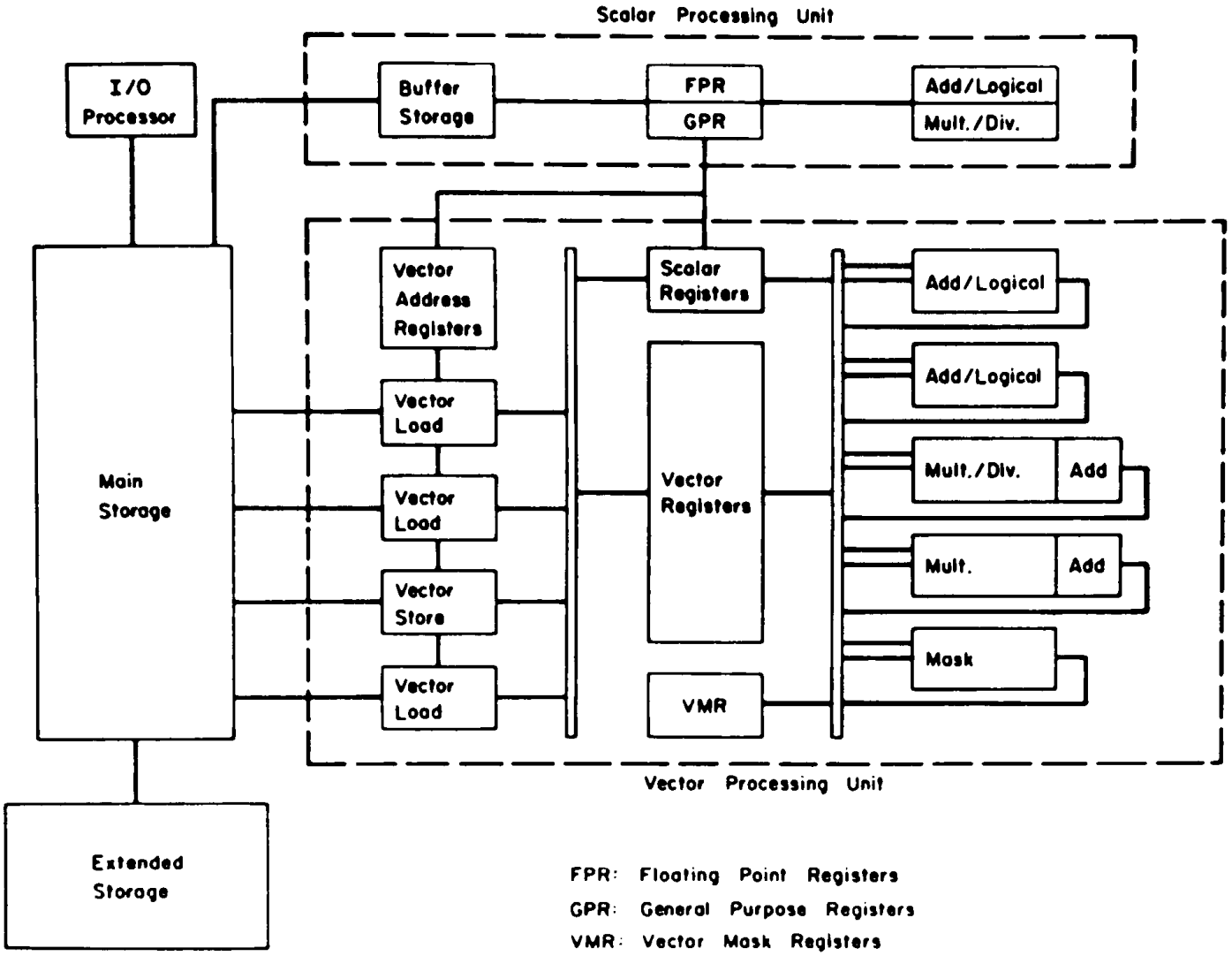

Figure 3
Hitachi S-810/10 Architecture

The Hitachi S-810/20 operates with either a batch operating system designed to interface with an IBM-compatible front-end computer via a channel-to-channel (CTC) adaptor in a loosely coupled network, or a stand-alone operating system with MVS-like batch and MVS/TSO-like interactive capabilities. The primary programming languages for the Hitachi S-810 computers are Fortran 77 and assembly language, although object code produced by any assembler or compiler available for IBM-compatible computers will also execute on the S-810 computers in scalar mode.

3. Comparison of Computers

In this section, we compare the IBM compatibility of the Fujitsu and Hitachi computers and discuss the similarities and differences between the Fujitsu, Hitachi, and CRAY X-MP-4 computers with regard to main storage, memory address architecture, memory, I/O, and vector and scalar processing performance.

3.1 IBM Compatibility of the Fujitsu and Hitachi Machines

Both Japanese computers run the full IBM System 370 scalar instruction set. The Japanese operating systems simulate IBM MVS system functions at the SVC level. MVS load modules created on Argonne's IBM 3033 ran correctly on both the Fujitsu and Hitachi machines in scalar mode.

The Japanese computers can share datasets on direct-access I/O equipment with IBM-compatible front-end machines. Codes can be developed and debugged on the front end with the user's favorite tools, then recompiled and executed on the vector processor. All software tools for the vector processor will run on the front end. Currently the software tools are MVS TSO/SPF oriented.

The Japanese Fortran compilers are compatible with IBM VS/Fortran.

3.2 Main Storage Characteristics

The main storage characteristics of the three machines in this study are compared in Table 2. All three machines have large, interleaved main memories, optimized for 64-bit-word data transfers, with bandwidths matched to the requirements of their respective vector units. Each machine permits vector accesses from contiguous, constant-stride separated, and scattered (using indirect list-vectors) memory addresses. All three machines use similar memory error-detection and error-correction schemes. The text that follows concentrates on those differences in main memory that have significant performance implications.

The CRAY X-MP-48 uses extremely fast bipolar memory, while the Fujitsu and Hitachi computers use relatively slower static-MOS memory (see Table 2). CRAY's choice of the faster but much more expensive bipolar memory is largely dictated by the need to service four processors from a single, symmetrically shared main memory. Fujitsu and Hitachi selected static MOS for its relatively lower cost and lower heat dissipation. These MOS characteristics permit much larger memory configurations without drastic cost and cooling penalties. Fujitsu and Hitachi compensate for the relatively slower speed of their MOS memory by providing much higher levels of memory banking and interleaving.

Table 1
Overview of Machine Characteristics

Characteristic	CRAY X-MP-4	Fujitsu VP-200	Hitachi S-810/20
Number of Processors	4	1	1
Machine Cycle Time	9.5 ns vector 9.5 ns scalar	7.5 ns vector 15 ns scalar	14 ns vector 28 ns scalar
Memory Addressing	Real	Mod. Virtual	Mod. Virtual
Maximum Memory Size	16 Mwords	32 Mwords	32 Mwords
Optional SSD Memory	32; 128 Mwords	Not Available	32; 64; 128 Mwords
SSD Transfer Rate	256 Mwords/s	Not Available	128 Mwords/s
I/O-Memory Bandwidth	50 Mwords/s	12 Mwords/s	12 Mwords/s
	(numbers below are per processor)		
CPU Memory Bandwidth	315 Mwords/s	533 Mwords/s	560 Mwords/s
Scalar Buffer Memory	64 Words T reg	8192 Words Cache	32768 Words Cache
Vector Registers	512 Words	8192 Words	8192 Words
Vector Pipelines:			
Load/Store Pipes	2 Load; 1 Store	2 Load/Store	3 Load; 1 Load/Store
Floating Point M & A	1 Mult; 1 Add;	1 Mult; 1 Add	2 Add; 2 Mult/Add
Peak Vector (M + A)	210 MFLOPS	533 MFLOPS	840 MFLOPS
Cooling System Type	Freon	Forced Air	Air and Radiator

Characteristic	CRAY X-MP-4	Fujitsu VP-200	Hitachi S-810/20
Operating Systems	CRAY-OS (batch) CTSS (interactive)	VSP (batch)	HAP OS
Front Ends	IBM, CDC, DEC, Data General, Univac, Apollo, Honeywell	IBM-compatible	IBM-compatible
Vectorizing Languages	Fortran 77	Fortran 77	Fortran 77
Other High-Level Languages	Pascal, C, LISP	Any IBM-compat.	Any IBM-compat.
Vectorizing Tools	Fortran Compiler	Fortran Compiler FORTUNE Interact. Vectorizer	Fortran Compiler VECTIZER

3.3 Memory Address Architecture

3.3.1 Memory Address Word and Address Space

The CRAY X-MP uses a 24-bit address, which it interprets as a 16-bit "parcel" address when referencing instructions and as a 64-bit-word address when referencing operands. This addressing duality leads to a 4-million-word address space for instructions and a 16-million-word address space for operands.

The Japanese machines use similar memory addressing schemes, owing to their mutual commitment to IBM compatibility. The two Japanese computers allow operating-system selection of IBM 370-compatible 24-bit addressing or IBM XA-compatible 31-bit addressing. These addressing alternatives provide a 2-million-word address space or a 256-million-word address space, respectively. The address space is identical for both program instructions and operands.

Table 2
Main Storage Characteristics

Memory Item	Units	CRAY X-MP-4	Fujitsu VP-200	Hitachi S-810/20
Memory Type	SECDED	16K-bit Bipolar	64K-bit S-MOS	64K-bit S-MOS
Addressing:	Type	Extended Real	Mod. Virtual	Mod. Virtual
Paged		No	System Only	System Only
Address Word	Bits	24	24 or 31	24 or 31
Address Space	Mwords	4(inst); 16(data)	2; 256	2; 256
Address Boundary:				
Instructions	Bit	16	16	16
Scalar Data	Bit	64	8	8
Vector Data	Bit	64	32; 64	32; 64
Vector Addressing		Contiguous	Contiguous	Contiguous
Modes		Constant Stride	Constant Stride	Constant Stride
		Indirect Index	Indirect Index	Indirect Index
Memory Size	Mwords	8; 16	8; 16; 32	4; 8; 16; 32
	Mbytes	64; 128	64; 128; 256	32; 64; 128; 256
Interleave	Sections	4; 4	8; 8; 8	8
	Ways	64; 64	128; 256; 256	128
Cycle Time:				
Section	CP - ns	1CP - 9.5 ns	2CP - 15 ns	1CP - 14 ns
Bank	CP - ns	4CP - 38 ns	8CP - 60 ns	5CP - 70 ns
Access Time:			From Cache	From Cache
Scalar	CP - ns	14CP - 133 ns	2CP - 15 ns	2CP - 28 ns
Vector	CP - ns	17CP - 162 ns	?	?

Memory Item	Units	CRAY X-MP-4	Fujitsu VP-200	Hitachi S-810/20
Transfer Rate:		(per CPU)		
Scalar L/S	Words/CP	1W/19. ns	2W/15 ns	2W/14 ns
Inst. Fetch	Words/CP	8W/9.5 ns	2W/15 ns	1W/14 ns
Vect. Load	Words/CP	2W/9.5 ns	8W/15 ns	8W/14 ns
Vect. Store	Words/CP	1W/9.5 ns	8W/15 ns	2W/14 ns
Vect. Total	Words/CP	3W/9.5 ns	8W/15 ns	8W/14 ns
I/O	Words/CP	1W/9.5 ns	?	1W/14 ns
Vector Bandwidth:		(per CPU)		
L/S Pipes	Pipes	2 Load; 1 Store	2 Load/Store	3 Load; 1 Load/Store
# Sectors	Sectors		x 2 Sectors	x 2 Sectors
Vector Bandwidth:	Stride	one; odd; even	one; odd; even	one; odd; even
Max. Load	Mwords/s	210; 210; 210	533; 266; 133	560; 560; 560
Max. Store	Mwords/s	105; 105; 105	533; 266; 133	140; 140; 140
Total L/S	Mwords/s	315; 315; 315	533; 266; 133	560; 560; 560
Scalar Buffer Memory:		T Registers	Cache Memory	Cache Memory
Size	Words	64 T	8192	32768
Block Load	Words/CP	1W/9.5 ns	8W/60 ns	8W/70 ns
Access Time	CP - ns	1CP - 9.5 ns	2CP - 15 ns	2CP - 28 ns
Trans. Rate	Words/CP	1W/9.5 ns	2W/15 ns	2W/28 ns
Instruction Buffer:		128 Words I-stack	Cache Memory	Cache Memory
Block Load	Words/CP	8W/9.5 ns	8W/60 ns	8W/70 ns

3.3.2 Operand Sizes and Operand Memory Boundary Alignment

CRAY X-MP computers have only two hardware operand sizes: 64-bit integer, real, and logical operands; and 24-bit integer operands, used primarily for addressing. All CRAY operands are stored in memory on 64-bit word boundaries. CRAY program instructions consist of one or two 16-bit "parcels," packed four to a word. CRAY instructions are fetched from memory, 32 parcels at a time beginning on an 8-word memory boundary, into an instruction buffer that in turn is addressable on 16-bit parcel boundaries.

The Japanese computers provide all of the IBM 370 architecture's operand types and lengths, and some additional ones. The Fujitsu and Hitachi scalar instruction sets can process 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit binary-arithmetic and logical operands; 8-bit to 128-bit (in units of 8 bits) decimal-arithmetic operands; and 8-bit to 32768-bit (in units of 8 bits) character operands. Scalar operands may be aligned in memory on any 8-bit boundary. However, the Fujitsu and Hitachi vector instruction sets can process only 32-bit and 64-bit binary-arithmetic and logical operands, and these operands must be aligned in memory on 32-bit and 64-bit boundaries, respectively. Most of the Fijitsu and Hitachi incompatibilities with IBM Fortran programs arise from vector operand misalignment in COMMON blocks and EQUIVALENCE statements.

3.3.3 Memory Regions and Program Relocation

The CRAY X-MP uses only real memory addresses. The operating system loads each program into a contiguous region of memory for instructions and a contiguous region of memory for operands. The CRAY X-MP uses two base registers to relocate all addresses in a program; one register uniformly biases all instruction addresses, and the second register uniformly biases all operand addresses.

In contrast, the Fujitsu and Hitachi computers use a modified virtual-memory addressing scheme. The operating systems and user application programs are each loaded into a contiguous region of "virtual" memory, although each may actually occupy noncontiguous "pages" of real memory. Every virtual address reference must undergo dynamic address translation to obtain the corresponding real memory address. As in conventional virtual-memory systems, operating-system pages can be paged out to an external device, allowing the virtual-memory space to exceed the underlying real-memory space. However, user application program pages are never paged out. Application program address translation is used primarily to avoid memory fragmentation.

3.3.4 Main Memory Size Limitations

The CRAY X-MP is available with up to 16 million words of main memory, the maximum permitted by its address space. This is restrictive compared to the Japanese offerings, especially as the memory must be shared by four processors. Currently, the Fujitsu and Hitachi computers offer a maximum of 32 million words of main memory. However, both Japanese computers could accommodate expansion to 256 million words (per program) within the current 31-bit virtual-addressing architecture.

3.4 Memory Performance

3.4.1 Memory Bank Structure

The computers on which we ran the benchmark problems were all equipped with 8 million words of main memory. The CRAY X-MP-48 memory is divided into 64 independent memory banks, organized as 4 sections of 16 banks each. Both the Fujitsu and Hitachi computer memories are divided into 128 independent memory banks organized as 8 sections of 16 banks each; Fujitsu memories larger than 8 million words have 256 memory banks in 8 sections. In general, the larger numbers of memory banks permit higher bandwidths for consecutive block memory transfers and fewer bank conflicts from random memory accesses.

3.4.2 Instruction Access

The CRAY X-MP has four 32-word instruction buffers that can deliver a new instruction for execution on every clock cycle, leaving the full memory bandwidth available for operand access. Each buffer contains 128 consecutive parcels of program instructions, but the separate buffers need not be from contiguous memory segments. Looping and branching within the buffers are permitted; entire Fortran DO loops and small subroutines can be completely contained in the buffer. An instruction buffer is block-loaded from memory, 32 words at a time, at the rate of 8 words per 9.5-nanosecond cycle.

The Fujitsu and Hitachi processors buffer all instruction fetches through their respective cache memories (see "Scalar Memory Access" below). The cache bandwidths are adequate to deliver instructions and scalar operands without conflict.

3.4.3 Scalar Memory Access

The CRAY X-MP does not have a scalar cache. Instead, it has 64 24-bit intermediate-address B-registers and 64 64-bit intermediate-scalar T-registers. These registers are under program control and can deliver one operand per 9.5-nanosecond clock cycle to the primary scalar registers. The user must plan a program carefully to make effective use of the B and T registers in CRAY Fortran; variables assigned to B and T registers by the compiler are never stored in memory.

The Fujitsu VP-200 and Hitachi S-810/20 automatically buffer all scalar memory accesses and instruction fetches through fast cache memories of 8192 words and 32768 words, respectively. The Fujitsu and Hitachi cache memories can each deliver two words per scalar clock cycle (15 nanoseconds and 28 nanoseconds, respectively) to their respective scalar execution units, entirely under hardware control.

3.4.4 Vector Memory Access

The computers studied all have multiple data-streaming pipelines to transfer operands between main memory and vector registers. Each processor of a CRAY X-MP has three pipelines – two dedicated to loads and one dedicated to stores – between its own set of vector registers and the shared main memory. (A fourth pipe in each X-MP processor is dedicated to I/O data transfers.) The Fujitsu VP-200 has two memory pipelines, each capable of both loads and stores. The Hitachi S-810/20 has four memory pipelines — three dedicated to loads and one capable of both loads and stores.

Each CRAY X-MP pipe can transfer one 64-bit word between main storage and a vector register each 9.5-nanosecond cycle, giving a single-processor memory bandwidth (excluding I/O) of 315 million words per second and a four-processor memory bandwidth of 1260 million words per second. The Fujitsu and Hitachi pipes can each transfer two 64-bit words each memory cycle (7.5 nanoseconds and 14 nanoseconds, respectively), giving total memory bandwidths of 533 and 560 million words per second, respectively.

For indirect-address operations (scatter/gather) and for constant strides different from one, the Fujitsu computer devotes one of its memory pipelines to generating operand addresses; its maximum memory-to-vector register bandwidth is 266 million words per second for scatter/gather and odd-number constant strides, and 133 million words per second for even-number constant strides.

All three machines can automatically "chain" their load and store pipelines with their vector functional pipelines. Thus, vector instructions need not wait for a vector load to complete, but can begin execution as soon as the first vector element arrives from memory. And vector stores can begin as soon as the first result is available in a vector register. In the limit, pipelines can be chained to create a continuous flow of operands from memory, through the vector functional unit(s), and back to memory with an unbroken stream of finished results. In this "memory-to-memory" processing mode, the vector registers serve as little more than buffers between memory and the functional units. The CRAY X-MP's three memory pipes permit memory-to-memory operation with two input operand streams and one result stream. With only two memory pipes, the Fujitsu VP-200 can function in memory-to-memory mode only if one of the input operands is already in a vector register, or if one of the operands is a scalar, and not at all if the vector stride is different from one. The Hitachi, with four memory pipes, can function in memory-to-memory mode with up to three input operand streams and one result stream; add to this the Hitachi's ability to automatically process vectors that are longer than its vector registers, and the Hitachi can be viewed as a formidable memory-to-memory processor.

3.5 Input/Output Performance

Table 3 summarizes the input/output features and performance of the CRAY X-MP, the Fujitsu, and the Hitachi. This information is entirely from the manufacturers' published machine specifications; no I/O performance comparisons were included in our tests.

Both the CRAY and Hitachi I/O subsystems have optional integrated solid-state storage devices, with data transfer rates of 2048 and 1024 Mbytes per second, respectively, over specialized channels. The I/O bandwidth of one of these devices dwarfs the I/O bandwidth of the entire disk I/O subsystem on each machine. The Fujitsu computers can attach only those solid-state storage devices that emulate standard IBM disk and drum devices over standard Fujitsu 3-Mbyte-per-second channels.

The IBM-compatible disk I/O subsystems on the two Japanese computers have a much larger aggregate disk storage capacity than the CRAY. The CRAY can attach a maximum of 32 disk units, while Fujitsu and Hitachi can each attach over one thousand disks. CRAY permits a maximum of 8 concurrent disk data transfers, while Fujitsu and Hitachi permit as many concurrent disk data transfers as there are channels (up to 31; at least one channel is required for front-end communication). Individually, CRAY's DD-49 disks can transfer data sequentially at the rate of 10 Mbytes per second, compared with only 3 Mbytes per second for the IBM 3380-compatible disks used by Fujitsu and Hitachi. But the maximum concurrent CRAY disk data rate (four DD-49 data streams on each of two I/O processors) is only 68 Mbytes per second, compared with 93 Mbytes per second for the two Japanese computers. The disks used on all three computers should have very similar random access performance, which is dominated by access time rather than data transfer rate.

CRAY includes up to 8 Mwords of I/O subsystem buffer memory between its CPUs and its disk units. This I/O buffer memory permits 100-Mbyte-per-second data transfer between the I/O subsystem and a single CRAY CPU. The IBM 3880-compatible disk controllers used by the two Japanese machines permit up to 2 Mwords of cache buffer memory on each controller. This disk controller cache does not increase peak data transfer rates but serves to reduce average record access times.

Table 3
Input/Output Features and Performance

I/O Features	CRAY X-MP-4	Fujitsu VP-200	Hitachi S-810/20
Disk I/O Channels:			
Disk I/O Processors	2 I/O Subsystems	2 I/O Directors	2 I/O Directors
Channels per IOP	1	16	16
Maximum Channels	2	32	32
Data Rate/Channel	100 MB/s	3 MB/s	3 MB/s
Total Bandwidth	200 MB/s	96 MB/s	96 MB/s
Disk Controllers:	DCU-5	6880	3880-equivalent
Max. per Channel	4	8	16
Max. Controllers	8	128	256
Disks/Controller	4	4-64	4-16
Data Paths/Controller	1	2	2
Bandwidth/Controller	12 MB/s	6 MB/s	6 MB/s
Disk Devices:	DD-39; DD-49	6380	3380-equivalent
Storage Capacity	1200 MB; 1200 MB	600 MB; 1200 MB	600 MB; 1200 MB
Data Transfer Rate	6 MB/s; 10 MB/s	3 MB/s	3 MB/s
Average Seek Time	18 ms; 16 ms	15 ms	15 ms
Average Latency	11 ms; ?	8 ms	8 ms
Maximum Striping	5; 3	24	?
Max. Disk Bandwidth	45 MB/s; 68 MB/s	93 MB/s	93 MB/s
Integrated SSD:	Optional	Not Available	Optional
Capacity (Mwords)	32; 64; 128		32; 64; 128
Data Transfer Rate	256 Mwords/s		128 Mwords/s

All three machines permit "disk striping" to increase I/O performance — the data blocks of a single file can be interleaved over multiple disk devices to allow concurrent data transfer for a single file. CRAY allows certain disks to be designated as striping volumes at the system level; striped and non-striped datasets may not reside on the same disk volume. A single CRAY file may be striped over a maximum of three DD-49 or five DD-39 disk units. Fujitsu and Hitachi permit striping on a dataset basis; striped and non-striped datasets may reside on the same disk volume. A single Fujitsu dataset may be striped over as many as 24 disk volumes.

3.6 Vector Processing Performance

Table 4 shows the vector architectures of the three computers studied. All three machines are vector register based, with multiple pipelines connecting the vector registers with main memory. All three have multiple vector functional units, permit concurrency among independent vector functional units and with the load/store pipelines, and permit flexible chaining of the vector functional units with each other and with the load/store pipelines. Although Fujitsu and Hitachi permit both 32-bit and 64-bit vector operands, all vector arithmetic on all three machines is performed in and optimized for 64-bit floating point. The three vector units differ primarily in the numbers and lengths of vector registers, the numbers of vector functional units, and the types of vector instructions.

Of the three machines, the CRAY has the smallest number and size of vector registers. Each CRAY X-MP processing unit has 8 vector registers of 64 elements, while the Fujitsu and Hitachi computers each have 8192-word vector register sets. The Fujitsu vector registers can be dynamically configured into different numbers and lengths of vector registers (see Table 4), ranging from a minimum of 8 registers of 1024 words each to a maximum of 256 registers of 32 words each. The Fujitsu Fortran compiler uses the vector-length information available at compile time to try to optimize the vector register configurations for each loop. The Hitachi has 32 vector registers, fixed at 256 elements each, but with the unique ability to process longer vectors without the user or the compiler dividing them into sections of 256 elements or less; the Hitachi hardware can automatically repeat a long vector instruction for successive vector segments. The HAP Fortran compiler decides when to divide vectors into 256-element segments and when to process entire vectors all at once, based on whether intermediate results in a vector register can be used in later operations.

Table 4
Vector Architecture

Vector Processing Item	CRAY X-MP/4	Fujitsu VP-200	Hitachi S-810/20
Vector Registers:			
Configuration	Fixed	Reconfigurable	Fixed
Total Capacity	512 Words/CPU	8192 Words	8192 Words
Number x Size	8x64 Words	8x1024 Words	32x256 Words
		16x512 Words	
		32x256 Words	
		64x128 Words	
		128x64 Words	
		256x32 Words	
Mask Registers	64 Bits	8192 Bits	8x256 Words
Vector Pipelines	(per CPU)		
Load/Store	2 Load; 1 Store	2 Load/Store	3 Load;1 Load/Store
Floating Point	1 Mult; 1 Add;	1 Mult; 1 Add	2 Add/Shift/Logic
	1 Recip. Approx.	1 Divide	1 Mult/Divide/Add
			1 Mult/Add
Other	1 Shift; 1 Mask	1 Mask	1 Mask
	2 Logical		
Maximum Vector Result Rates			
(64-bit results):			
Floating Point Mult.	105 MFLOPS	267 MFLOPS	280 MFLOPS
Floating Point Add	105 MFLOPS	267 MFLOPS	560 MFLOPS
Floating Point Divide	33 MFLOPS	56 MFLOPS	70 MFLOPS
Floating Mult. & Add	210 MFLOPS	533 MFLOPS	560 MFLOPS
			840 (M+2A)
Vector Data Types:			
Floating Point	64-bit	32-bit; 64-bit	32-bit; 64-bit
Fixed Point	64-bit	32-bit	32-bit
Logical	64-bit	1-bit; 64-bit	64-bit
Vector Macro Instructions:			
Masked Arithmetic	No	Yes	Yes
Vector Compress/Expand	Yes	Yes	Yes
Vector Merge under Mask	Yes	No	No
Vector Sum (S=S+Vi)	No	Yes	Yes

Vector Processing Item	CRAY X-MP-4	Fujitsu VP-200	Hitachi S-810/20
Vector Macro Instructions:			
Vector Prod (S=S*Vi)	No	No	Yes
DOT Product (S=S+Vi*Vj)	No	Chain	Yes
DAXPY (Vi=Vi+S*Xi)	Chain	Chain	Yes
Iteration (Aj=Ai*Bi+Ci)	No	No	Yes
Max/Min (S=MAX(S,Vi))	No	Yes	Yes
Fix/Float (Vi=Ii;Ii=Vi)	Chain	Yes	Yes

The Hitachi has more vector arithmetic pipelines than the CRAY and Fujitsu computers. These pipelines permit the Hitachi to achieve higher peak levels of concurrency than CRAY and Fujitsu. Depending on the operation mix, the Hitachi can drive two vector add and two vector multiply+add pipelines concurrently, for an instantaneous result rate of 840 MFLOPS. If the program operation mix is inappropriate, however, the extra pipelines are just expensive unused hardware. The HAP Fortran "pair-processing" option often increases performance by dividing a vector in two and processing each half concurrently through a separate pipe. For long vectors, pair-processing can double the result rate; but for short vectors, startup overhead can result in reduced performance. The HAP Fortran compiler pemits pair-processing to be selected on a program-wide, subroutine-wide, or individual loop basis.

The Fujitsu and Hitachi computers have larger and more powerful vector instruction sets than the CRAY. These macro instruction sets make these machines more "compilable" and more "vectorizable" than the CRAY. Especially valuable are the macro instructions that reduce an entire vector operation to a single result, such as the vector inner (or dot) product. The CRAY, lacking such instructions, must normally perform these operations in scalar mode, although vectorizable algorithms exist for long CRAY vectors. The Hitachi has the richest set of vector macro-instructions, with macro functional units to match. Both Fujitsu and Hitachi have single vector instructions or chains to extract the maximum and minimum elements of a vector, to sum the elements of a vector, to take the inner product of two vectors, and to convert vector elements between fixed point and floating point representations. To these, the Hitachi adds a vector product reduction, the DAXPY sequence common in linear algebra, and a vector iteration useful in finite-difference calculations.

The only CRAY masked vector instructions are the vector compress/expand and conditional vector merge instructions; the CRAY Fortran compiler uses these instructions to vectorize loops with only a single IF statement. The CRAY can hold logical data for only a single vector register. Both Japanese computers, on the other hand, have masked arithmetic instructions that permit straightforward vectorization of loops with IF statements. The Fujitsu and Hitachi computers have mask register sets that can hold logical data for every vector register element. These large mask register sets, and vector logical instructions to manipulate these masks, should make the Japanese machines strong candidates for logic programming. These machines can hold the results of many different logical operations in their multiple mask registers, eliminating the need to recompute masks that are needed repeatedly, and

permitting the vectorization of loops with multiple, compound, and nested IF statements.

3.7 Scalar Processing Performance

Table 5 compares the scalar architectures of the three machines studied.

All three computers permit scalar and vector instruction concurrency; CRAY permits concurrency among all its functional units. The Fujitsu and Hitachi computers are compatible with IBM System 370; they implement the complete IBM 370 scalar instruction set and scalar register sets (Fujitsu added four additional floating-point registers).

CRAY computers use multiple, fully-segmented functional units for both scalar and vector instruction execution, while Fujitsu and Hitachi use an unsegmented execution unit for all scalar instructions. CRAY computers can begin a scalar instruction on any clock cycle; more than one CRAY scalar instruction can be in execution at a given time, in the same and in different functional units. Fujitsu and Hitachi, on the other hand, perform their scalar instructions one at a time, many taking more than one cycle. Thus, even though many scalar instruction times are faster on the Fujitsu than on the CRAY, the CRAY will often have a higher scalar result rate because of concurrency. In our benchmark set, a single processor of the CRAY X-MP-4 outperformed both the Fujitsu VP-200 and the Hitachi S-810/20 on all the programs that were dominated by scalar floating point instruction execution.

The Fujitsu and Hitachi computers have larger and more powerful general-purpose instruction sets than the CRAY, and more flexible data formats for integer and character processing. Thus, applications that are predominately scalar and use little floating-point arithmetic may well execute faster on these IBM-compatible computers than on a CRAY. We had no applications in our benchmark to measure such performance.

Table 5
Scalar Architecture

Scalar Processing Item	CRAY X-MP-4	Fujitsu VP-200	Hitachi S-810/20
Scalar Cycle Time	9.5 nsec	15 nsec	28 nsec
Scalar Registers:			
General/Addressing	8x24-bit	16x32-bit	16x32-bit
Floating Point	8x64-bit	8x64-bit	4x64-bit
Scalar Buffer Memory:	T-Registers	Cache Memory	Cache Memory
Capacity	64 Words	8192 Words	32768 Words
Memory Bandwidth	105 Mwords/sec	266 Mwords/sec	112 Mwords/sec
CPU Access Time	1 CP - 9.5 nsec	1 CP - 15 nsec	1 CP - 28 nsec
CPU Transfer Rate	1 Word/9.5 nsec	2 Words/15 nsec	2 Words/28 nsec
Scalar Execution Times:			
Floating Point Mult.	7 CP - 66.5 nsec	3 CP - 45 nsec	3 CP - 84 nsec
Floating Point Add	6 CP - 57.0 nsec	2 CP - 30 nsec	2 CP - 56 nsec
Scalar Data Types:			
Floating Point	64-bit	32; 64; 128-bit	32; 64; 128-bit
Fixed Point	24; 64-bit	16; 32-bit	16; 32-bit
Logical	64-bit	8; 32; 64-bit	8; 32; 64-bit
Decimal	None	1 to 16-bytes	1 to 16-bytes
Character	None	1 to 4096-bytes	1 to 4096-bytes

4. Benchmark Environments

We spent two days at Cray Research compiling and running the benchmark on the CRAY X-MP-4. The CRAY programs were one-processor tests; no attempt was made to exploit the additional processors.

For the Japanese benchmarkings, we sent ahead a preliminary tape of our benchmark source programs and some load modules produced at Argonne. At both Fujitsu and Hitachi the load modules ran without problem, demonstrating that the machines are in fact compatible with IBM computers on both instruction set and operating system interface levels. (Of course, these tests did not use the vector features of the machines.)

The VP-200 tests were run at the Fujitsu plant in Numazu, Japan, during a one-week period. We had as much time on the VP-200 as needed. The front-end machine was a Fujitsu M-380 (approximately twice as fast as a single processor of an IBM 3081 K).

The Hitachi S-810/20 tests were run at the Hitachi Kanagawa Works, during two afternoons. The Hitachi S-810/20 benchmark configuration had no front-end system. Instead, we compiled, linked, ran, and printed output directly on the machine.

The physical environment of the Hitachi S-810/20 at Kanagawa is noteworthy. The machine room was not air-conditioned; a window was opened to cool off the area. The outside temperature exceeded 100 degrees Fahrenheit on the first day, and we estimate that the computer room temperature was well above 100 degrees, with high humidity; yet the computer ran without problem.

5. Benchmark Codes and Results

5.1 Codes

We asked some of the major computer users at Argonne for typical Fortran programs that would help in judging the performance of these vector machines. We gathered 20 programs, some simple kernels, others full production codes. The programs are itemized in Table 6.

Four of the programs have very little vectorizable Fortran (for the most part they are scalar programs): BANDED, NODAL0, NODAL1, SPARSESP. Both STRAWEXP and STRAWIMP have many calculations involving short vectors. For most of these programs the CRAY X-MP performed fastest, with the Fujitsu faster than the Hitachi.

Below we describe some of the benchmarks and analyze the results.

5.1.1 APW

The APW program is a solid-state quantum mechanics electronic structure code. APW calculates self-consistent field wave functions and energy band structures for a sodium chloride lattice using an antisymmetrized plane wave basis set and a muffin-tin potential. The majority of loops in this program are short and are coded as IF loops rather than DO loops; they do not vectorize on any of the benchmarked computers. The calculations are predominately scalar.

This program highlights the CRAY X-MP advantage when executing "quasi-vector" code (vector-like loops that do not vectorize for some reason). The CRAY executes scalar code on segmented functional units and can achieve a higher degree of concurrency in scalar than either the Fujitsu or Hitachi machines, which execute scalar instructions one at a time.

5.1.2 BIGMAIN

BIGMAIN is a highly vectorized Monte Carlo algorithm for computing Wilson line observables in SU(2) lattice gauge theory. This program has the longest vector lengths of the benchmarks. All the vectors begin on the same memory bank boundary, and all have a stride of twelve. The only significant nonvectorized code is an IF loop, which seriously limits the peak performance.

The superior performance of the CRAY on BIGMAIN reflects both the CRAY's insensitivity to the vector stride and its greater levels of concurrency when executing scalar loops. The Fujitsu performance reflects a quartering of memory bandwidth when using a vector stride of twelve. The Hitachi performance reflects its slower scalar performance.

5.1.3 BFAUCET and FFAUCET

BFAUCET and FFAUCET compute the ground state energies of drops of liquid helium by the variational Monte Carlo method. The BFAUCET codes involve Bose statistics, and a table-lookup operation is an important component of the time. The FFAUCET cases use Fermi statistics and are dominated by the evaluation of determinants using LU decomposition. The different cases correspond to different sized drops, as shown in Table 7.

BFAUCET1, 2, and 3 and FFAUCET1 and 2 perform only a single Monte Carlo iteration each; these cases are typical of checkout runs and are dominated by non-repeated setup work. BFAUCET4, 5, and 6 and FFAUCET3 are long production runs.

5.1.4 LINPACK

The LINPACK timing is dominated by memory reference as a result of array access through the calls to SAXPY. For this problem the vector length changes during the calculation from length 100 down to length 1 (see Table 8).

Fujitsu's and Hitachi's performance reflects the fact that they do not do so well as the CRAY with short vectors.

Table 6
Programs Used for Benchmarking

Code	No. of Lines	Description
APW	1448	Solid-state code, for anti-symmetric plane wave calculations for solids.
BANDED	1539	Band linear algebra equation solver, for parallel processors.
BIGMAIN	774	Vectorized Monte Carlo algorithm, for SU(2) lattice gauge theory.
DIF3D	527	1, 2, and 3-D diffusion theory kernels.
LATFERM3	1149	Statistical-mechanical approach to lattice gauge calculations.
LATFERM4	1149	Statistical-mechanical approach to lattice gauge calculations.
LATTICE8	1149	Statistical-mechanical approach to lattice gauge calculations.
MOLECDYN	1020	Molecular dynamics code simulating a fluid.
NODAL0	345	Kernel of 3-D neutronics code using nodal method.
NODAL1	345	Kernel of 3-D neutronics code using nodal method.
NODALX	345	Kernel of 3-D neutronics code using nodal method.
BFAUCET	5460	Variational Monte Carlo for drops of He-4 atoms — Bose statistics.
FFAUCET	5577	Variational Monte Carlo for drops of He-3 atoms — Fermi statistics.
SPARSESP	1617	ICCG for non-symmetric sparse matrices based on normal equations.
SPARSE1	3228	MA32 from the Harwell library sparse matrix code using frontal techniques and software run on a 64 x 64 problem.
STRAWEXP	4806	2-D nonlinear explicit solution of finite element program with weakly coupled thermomechanical formulation in addition to structural and fluid structural interaction capability.
STRAWIMP	4806	Same as STRAWEXP but implicit solution.

Table 7
Average Vector Length for BFAUCET and FFAUCET

Case	Average Vector Length
BFAUCET1	10
BFAUCET2	35
BFAUCET3	56
BFAUCET4	120
BFAUCET5	10
BFAUCET6	35
FFAUCET1	10
FFAUCET2	17
FFAUCET3	10

Table 8
LINPACK Timing for a Matrix of Order 100

Machine	MFLOPS	Seconds
CRAY X-MP	21	.032
Fujitsu VP-200	17	.040
Hitachi S-810/20	17	.042

5.1.5 LU, Cholesky Decomposition, and Matrix Multiply

The LU, Cholesky decomposition, and matrix multiply benchmarks are based on matrix vector operations. As a result, memory reference is not a limiting factor since results are retained in vector registers during the operation. The technique used in these tests is based on vector unrolling [1], which works equally well on CRAY, Fujitsu, and Hitachi machines.

The routines used in Tables 9 through 11 have a very high percentage of floating-point arithmetic operations. The algorithms are all based on column accesses to the matrices. That is, the programs reference array elements sequentially down a column, not across a row. With the exception of matrix multiply, the vector lengths start out as the order of the matrix and decrease during the course of the computation to a vector length of one.

Table 9
LU Decomposition Based on Matrix Vector Operations

	MFLOPS		
Order	CRAY X-MP (1 CPU)	Fujitsu VP-200	Hitachi S-810/20
50	24.5	20.5	17.9
100	51.6	51.8	47.5
150	72.1	84.6	76.3
200	87.4	117.1	102.2
250	99.2	148.8	126.4
300	108.4	178.8	147.8

Table 10
Cholesky Decomposition Based on Matrix Vector Operations

	MFLOPS		
Order	CRAY X-MP (1 CPU)	Fujitsu VP-200	Hitachi S-810/20
50	29.9	25.8	18.8
100	65.6	70.6	60.1
150	91.9	117.6	104.9
200	107.7	162.2	144.9
250	119.1	202.2	179.7
300	132.3	238.1	211.8

Table 11
Matrix Multiply Based on Matrix Vector Operations

	MFLOPS		
Order	CRAY X-MP (1 CPU)	Fujitsu VP-200	Hitachi S-810/20
50	98.4	112.9	100.0
100	135.7	225.2	213.3
150	149.0	328.1	279.3
200	156.2	404.5	336.8
250	165.9	462.2	366.7
300	167.9	469.2	390.4

For low-order problems the CRAY X-MP is slightly faster than the VP-200 and S-810/20, because it has the smallest vector startup overhead (primarily due to faster memory access). As the order increases, and the calculations become saturated by longer vectors, the Fujitsu VP-200 attains the fastest overall execution rate.

With matrix multiply, the vectors remain the same length throughout; here Fujitsu comes close to attaining its peak theoretical speed in Fortran.

5.2 Results

Table 12 contains the timing data for our benchmark codes. We also include the timing results on other machines for comparison.

6. Fortran Compilers and Tools

6.1 Fortran Compilers

The three compilers tested exhibit several similarities. All three tested systems include a full Fortran 77 vectorizing compiler as the primary programming language. The CRAY compiler includes most IBM and CDC Fortran extensions; the two Japanese compilers include all the IBM extensions to Fortran 77. All three compilers can generate vectorized code from standard Fortran; no explicit vector syntax is provided. All three compilers recognize a variety of compiler directives — special Fortran comments that, when placed in a Fortran source code, aid the compiler in optimizing and vectorizing the generated code. Each compiler, in its options and compiler directives, provides users with a great deal of control over the optimization and vectorization of their programs.

Table 12
Timing Data (in seconds) for Various Computers (a)

Program Name	CRAY X-MP-4 using 1 proc.	Fujitsu VP-200	Hitachi S810/20	Hitachi(b) S810/20	Hitachi(b) S810/20	IBM 370/195	IBM 3033	IBM 3033	Amdahl 5860
	CFT 1.13	f77	f77	FORTVS (scalar)	H EXT (scalar)	H EXT	FORTVS	H EXT	f77
APW	**30.69**	**40.58**	**54.37**				171		62
BANDED	**24.3**	**34.15**	**38.3**	**41.0**			102.65		35
BIGMAIN	**10.86**	**23.49**	**34.36**		157.66				
DIF3DS1/1	**23.7**	**20.31**	**21.9**	**45.1**	**39.2**	**74.82**	**151.81**	**134.2**	**62**
DIF3DS2/1	**19.0**	**21.93**	**21.9**	**47.4**	**41.5**	**81.27**	**157.44**	**142.**	**67**
DIF3DV0/1	**9.31**	**16.37**	**11.8**	**50.1**	**39.5**	**73**	**168**	**138**	**73**
DIF3DV1/1	**9.37**	**16.59**	**12.1**	**49.3**	**38.7**	**74**	**167**	**137**	**70**
LATFERM3	**6.1**	**6.2**	**6.6**	**15.8**	**33.3**		**52.07**	**87.8**	**18**
LATFERM4	**121.8**	**65.29**	**65.3**	**345.2**	**820.6**				**640**
LATTICE8	**10.2**	**5.54**	**6.7**	**16**	**19.4**		**46.38**	**53.8**	**17**
MOLECDYN	**8.68**	**9.07**	**15.78**	**16.6**	**17.2**	**36.26**	**51.44**	**51.74**	**17**
NODAL0	**6.41**	**14.31**	**20.1**	**19.5**	**19.7**	**28.36**	**45.53**	**45.5**	**27**
NODAL1	**6.45**	**14.47**	**19.8**	**19.3**	**19.5**	**27.58**	**45.35**	**45.**	**23**
NODELX	**.25**	**.14**	**.20**		**1.14**	**1.45**	**1.57**		
BFAUCET1	**11.2**	**16.13**		**22.9**	**22.8**		**74**	**73**	**31**
BFAUCET2	**8.96**	**11.66**		**23.9**	**24.2**		**79**	**78**	**34**
BFAUCET3	**10.6**	**18.48**		**38.7**	**38.9**		**130**	**128**	**405**
BFAUCET4	**259.4**	**551.2**			**621.0**		**2100**	**2048**	**920**
BFAUCET5	**787.4**	**923.04**			**1529.4**				**2351**
BFAUCET6	**727.5**	**823.98**							**2786**
FFAUCET1	**13.6**	**19.45**			**26.7**		**94**	**82**	**35**
FFAUCET2	**44.4**	**42.31**			**114.3**		**419**	**397**	**150**
FFAUCET3	**1144.0**	**1691.83**							**2440**
SPARSESP	**1200**	**1361**	**1264.29**						**1484**
SPARSE1	**2.51**	**6.74**	**9.85**	**14.26**			**33.06**		**26**
STRAWEXP	**37.3**	**45.74**	**59.2**			**116.28**	**143.35**	**142.28**	**51**
STREWEXP2	**153.4**	**179.37**	**231.13**		**273.9**				**216**
STRAWIMP	**151.5**	**151.51**	**172.61**		**?**	**382.73**	**381.51**	**360.55**	

(a) Numbers in boldface denote "fastest" time for a given program.

(b) From load modules created on an IBM machine.

The compilers differ primarily in the range of Fortran statements they can vectorize, the complexity of the DO loops that they vectorize, and the quantity and quality of messages they provide the programmer about the success or failure of vectorization.

All three Fortran compilers have similar capabilities for vectorizing simple inner DO loops and DO loops with a single IF statement. The two Japanese compilers can also vectorize outer DO loops and loops with compound, multiple, and nested IF statements. The Fujitsu compiler has multiple strategies for vectorizing DO loops containing IF statements, based on compiler directive estimates of the IF statement true ratio. The Japanese compilers can vectorize loops that contain a mix of vectorizable and non-vectorizable statements; the CRAY compiler requires the user to divide such code into separate vectorizable and non-vectorizable DO loops.

The vector macro instructions (e.g., inner product, MAX/MIN, iteration) on the two Japanese computers permit their compilers to vectorize a wider range of Fortran statements than can the CRAY compiler. And, the Japanese compilers seem more successful at using information from outside a DO loop in determining whether that loop is vectorizable.

All three compilers, in their output listings, indicate which DO loops vectorized and which did not. The two Japanese compilers provide more detailed explanations of why a particular DO loop or statement does not vectorize. The Fujitsu compiler listing is the most effective of the three: in addition to the vectorization commentary, the Fujitsu compiler labels each DO statement in the source listing with a V if it vectorizes totally, an S if the loop compiles to scalar code, and an M if the loop is a mix of scalar and vector code. Each statement in the loop itself is similarly labeled.

The Fujitsu and Hitachi compilers make all architectural features of their respective machines available from standard Fortran. As a measure of confidence in their compilers, Fujitsu has written all, and Hitachi nearly all, of their scientific subroutine libraries in standard Fortran.

6.2 Fortran Tools

Fujitsu and Hitachi provide Fortran source program analysis tools which aid the user in optimizing program performance. The Fujitsu interactive vectorizer is a powerful tool for both the novice and the experienced user; it allows one to tune a program despite an unfamiliarity with vector machine architecture and programming practices. The interactive vectorizer (which runs on any IBM-compatible system with MVS/TSO) displays the Fortran source with each statement labeled with a V (vectorized), S (scalar), or M (partially vectorized), and a static estimate of the execution cost of the statement. As the user interactively modifies a code, the vectorization labels and statement execution costs are updated on-screen. The vectorizer gives detailed explanations for failure to vectorize a statement, suggests alternative codings that will vectorize, and inserts compiler directives into the source based on user responses to the vectorizer's queries. Statement execution cost analyses are based on assumed DO loop iteration counts and IF statement true ratios. The user can run the FORTUNE execution analyzer to gather run-time statistics for a program, which can then be input to the interactive vectorizer to provide a more accurate dynamic statement execution cost analysis.

The Hitachi VECTIZER runs in batch mode; it provides additional information much like the Hitachi Fortran compiler's vectorization messages.

7. Conclusions

The results of our benchmark show the CRAY X-MP-4 to be a consistently strong performer across a wide range of problems. The CRAY was particularly fast on programs dominated by scalar calculations and short vectors. The fast CRAY memory contributes to low vector startup times, leading to its exceptional short-vector performance. The CRAY scalar performance derives from its segmented functional units; the X-MP achieves enough concurrency in many scalar loops to outperform the Japanese machines, even though individual scalar arithmetic instruction times are about twice as long on the CRAY as on the Fujitsu.

The Fujitsu and Hitachi computers perform faster than the CRAY for highly vectorizable programs, especially those with long (>50) vector lengths. The Fujitsu VP achieved the most dramatic peak performance in the benchmark, outperforming a single CRAY X-MP processor by factors of two to three on matrix-vector algorithms, with the Hitachi not far behind. Over the life cycle of a program, the Fujitsu and Hitachi machines should benefit relatively more than the CRAY from tuning that increases the degree of program vectorization.

The CRAY has I/O weaknesses that were not probed in this exercise. With an SSD, the CRAY has the highest I/O bandwidth of the three machines. However, owing to severe limits on the number of disk I/O paths and disk devices, the total CRAY disk storage capacity and aggregate disk I/O bandwidth fall far below that of the two Japanese machines. The CRAY is forced to depend on a front-end machine's mass storage system to manage the large quantities of disk data created and consumed by such a high-performance machine.

Several weaknesses were evident in the Fujitsu VP in this benchmark. The Fujitsu memory performance degrades seriously for nonconsecutive vectors. This was particularly evident in the BIGMAIN, DIF3D, and FAUCET benchmark programs. Even-number vector strides reduce the Fujitsu memory bandwidth by 75%, and a stride proportional to the number of memory banks (stride=n*128) reduces the memory bandwidth about 94%. This results in poor performance for vectorized Fortran COMPLEX arithmetic (stride=2). Fujitsu users will profit by reprogramming their complex arithmetic using only REAL arrays, and by ensuring that multidimensional-array algorithms are vectorized by column (stride=1) rather than by row.

Fujitsu's vector performance is substantially improved if a program's maximum vector lengths are evident at compile time, whether from explicit DO loop bounds, array dimension statements, or compiler directives. For example, the order-100 LINPACK benchmark improves by 12% to 19 MFLOPS, and the order-300 matrix-vector LU benchmark improves by 23% to 220 MFLOPS, when a Fujitsu compiler directive is included to specify the maximum vector length (numbers from the LINPACK benchmark paper [2]). When maximum vector lengths are known, the Fujitsu compiler can optimize the numbers and lengths of the vector registers and frequently avoid the logic that divides vectors into segments no larger than the vector registers. Fujitsu's short-loop performance, not strong

to begin with, is particularly degraded by unnecessary vector segmentation ("stripmining") logic. None of the benchmark problems had explicit vector length information.

In many ways, the Hitachi computer seems to have the greatest vector potential. Despite its slower memory technology, the Hitachi has the highest single processor memory bandwidth, owing to its four memory pipes. Also, Hitachi has the most powerful vector macro instruction set and the most flexible set of arithmetic pipelines; in addition, the Hitachi is the only computer able to process vectors longer than its vector registers, entirely in hardware. The vectorizing Fortran compiler is impressive, although the compiler is rarely able to exploit fully the potential concurrency of the arithmetic pipelines. The Hitachi performs best on the benchmarks with little scalar content; its slow scalar performance — about half that of the Fujitsu computer — burdens its performance on every problem.

At present the Japanese Fortran compilers are superior to the CRAY compiler at vectorization. Advanced Fujitsu and Hitachi hardware features provide opportunities for vectorization that are unavailable on the CRAY. For example, the Japanese machines have macro instructions to vectorize dot products, simple recurrences, and the search for the maximum and minimum elements of an array; and they have multiple mask registers to allow vectorization of loops with nested IF statements. Thus, a wider range of algorithms can vectorize on the Japanese computers than can vectorize on the CRAY. Also, the Japanese compilers provide the user with more useful information about the success and failure of vectorization. Moreover, there is no CRAY equivalent to the Fujitsu interactive vectorizer and FORTUNE performance analyzer. These advanced hardware features and vectorizing tools will make it easier to tune programs for optimum performance on the Japanese computers than on the CRAY.

The CRAY X-MP and the Japanese computers require different tuning strategies. The CRAY compiler does not partially vectorize loops. Therefore, CRAY users typically break up loops into their vectorizable and nonvectorizable parts. The Japanese compilers, however, automatically segment loops into their vectorizable and nonvectorizable parts.

References

[1] J.J. Dongarra and S.C. Eisenstat, "Squeezing the Most out of an Algorithm in CRAY Fortran," *ACM Trans. Math. Software*, Vol. 10, No. 3, pp. 221-230 (1984).

[2] J. J. Dongarra, *Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment*, Argonne National Laboratory Report MCS-TM-23 (October 1985)

Acknowledgment

We would like to thank Gail Pieper for her excellent help in editing this report.

Appendix A: VECTOR Program

Below is the program VECTOR, used to check out the compiler's ability to vectorize Fortran code.

VECTOR is not a particularly difficult benchmark for vectorizing compilers, but there are a number of tricky loops and the program that will identify poor vector compilers. Each loop is designed to test the ability of the compiler to detect a single opportunity for vectorization. By no means is it intended to be an exhaustive test.

```
C      PROGRAM VECT(INPUT,OUTPUT)
       INTEGER LOOP, PRTINC
C
       REAL START, MAXNUM, MINFRC
C
       INTEGER N01, N02, N03, N04, N05, N06, N07, N08, N09, N10,
      .        N11, N12, N13, N14, N15, N16, N17
C
       LOGICAL ADD, SUB, MULT, DIV
C
       INTEGER SIZE01, SIZE02, SIZE03, SIZE04, SIZE05,
      .        SIZE06, SIZE07, SIZE08, SIZE09, SIZE10,
      .        SIZE11, SIZE12, SIZE13, SIZE14, SIZE15,
      .        SIZE16, SIZE17, SIZE99
C
       INTEGER SZ16SQ, SZ17SQ
C
C  ALL PARAMETER STATEMENTS FOLLOW
C
C
C
C  THE FOLLOWING PARAMETER, LOOP, CONTROLS THE NUMBER OF TIMES
C  THE MAJOR LOOP IS EXECUTED.  ONE EXECUTION OF THE MAJOR LOOP
C  CAUSES ALL OF THE 17 MINOR LOOPS TO BE EXECUTED ONCE.
C
       PARAMETER ( LOOP = 10000)
C
C  THE FOLLOWING PARAMETER, PRTINC, CONTROLS THE AMOUNT OF OUTPUT
C
       PARAMETER ( PRTINC = 10 )
C
       PARAMETER ( START = 1.01, MAXNUM = 1.E50, MINFRC = 1./MAXNUM )
C
       PARAMETER ( ADD  = .TRUE., SUB = .FALSE.,
      .            MULT = .TRUE., DIV = .FALSE. )
C
C
C  THE FOLLOWING SIZE PARAMETERS MAY BE FREELY CHANGED BY THE USER.
C  IT MAY BE DESIRABLE TO HAVE A MIXTURE OF LARGE AND SMALL ARRAYS.
C
C  ALL MATRICES ARE SQUARE AND THE SIZE PARAMETER IS THE NUMBER OF
C  ROWS (OR COLUMNS) IN THE MATRIX, NOT THE TOTAL NUMBER OF ELEMENTS.
```

```
C  THE COMPUTATION SIZE*SIZE TO DETERMINE THE NUMBER OF ELEMENTS.
C  MATRICES ARE USED IN LOOPS 16 (SIZE16) AND 17 (SIZE17).
C
      PARAMETER (
     .  SIZE01 =  100, SIZE02 = 1000,
     .  SIZE03 =  100, SIZE04 = 6000,
     .  SIZE05 = 1000, SIZE06 = 1000,
     .  SIZE07 = 1000, SIZE08 = 1000,
     .  SIZE09 = 1000, SIZE10 = 1000,
     .  SIZE11 = 1000, SIZE12 = 4000,
     .  SIZE13 = 1000, SIZE14 = 1000,
     .  SIZE15 = 1000, SIZE16 = 100,
     .  SIZE17 = 100                    )
C
      PARAMETER (
     .  SZ16SQ = SIZE16*SIZE16, SZ17SQ = SIZE17*SIZE17  )
C
C
C  THE SIZE OF THE '99' ARRAYS IS DEFINED TO BE THE LARGEST SIZE
C  OF ALL THE SINGLE DIMENSION ARRAYS.
C
      PARAMETER ( SIZE99 = SIZE04 )
C
C  THE LOOP MAXIMUMS ARE DEFINED TO BE THE SIZE OF THE ARRAY
C      '$
C  THAT ARE USED IN THE LOOP.
C
      PARAMETER (
     .  N01 = SIZE01, N02 = SIZE02,
     .  N03 = SIZE03, N04 = SIZE04,
     .  N05 = SIZE05, N06 = SIZE06,
     .  N07 = SIZE07, N08 = SIZE08,
     .  N09 = SIZE09, N10 = SIZE10,
     .  N11 = SIZE11, N12 = SIZE12,
     .  N13 = SIZE13, N14 = SIZE14,
     .  N15 = SIZE15, N16 = SIZE16,
     .  N17 = SIZE17                )
C
C  THE REAL ARRAY DECLARATION STATEMENTS FOLLOW
C
      REAL V01A(SIZE01), V01B(SIZE01), V02A(SIZE02), V02B(SIZE02),
     .     V03A(SIZE03), V04A(SIZE04), V05A(SIZE05), V06A(SIZE06),
     .     V07A(SIZE07), V08A(SIZE08), V08B(SIZE08), V09A(SIZE09),
     .     V10A(SIZE10), V11A(SIZE11), V12A(SIZE12), V12B(SIZE12),
     .     V12C(SIZE12), V13A(SIZE13), V13B(SIZE13), V14A(SIZE14),
     .     V15A(SIZE15)
      REAL V99A(SIZE99), V99B(SIZE99), V99C(SIZE99)
      REAL M16A(SIZE16,SIZE16), M16B(SIZE16,SIZE16),
     .     M17A(SIZE17,SIZE17)
C
C  EACH INTEGER ARRAY IS USED AS AN INDEX INTO A REAL ARRAY.
C  ARRAY I<NAME> IS USED AS AN INDEX INTO ARRAY V<NAME>.
C  THEREFORE, THE SIZE OF ARRAY I<NAME> IS MADE THE SAME
C  AS ARRAY V<NAME>.
C
C
      INTEGER I15A(SIZE15), I99A(SIZE99), I99B(SIZE99), I99C(SIZE99)
C
C
```

```
C  ALL SCALAR VARIABLES ARE DECLARED
C
      REAL S, T, X
      INTEGER I, J, K, M, INDEX
      LOGICAL OP01, OP02, OP03, OP08, OP12, OP13, OP16, OP17
C-----------------------------------------------------------
C  INITIALIZE ALL VARIABLES
C-----------------------------------------------------------
      DATA
     .  V01A /SIZE01 * START/, V01B /SIZE01 * START/,
     .  V02A /SIZE02 * START/, V02B /SIZE02 * START/,
     .  V03A /SIZE03 * START/, V04A /SIZE04 * START/,
     .  V05A /SIZE05 * START/, V06A /SIZE06 * START/,
     .  V07A /SIZE07 * START/, V08A /SIZE08 * START/,
     .  V08B /SIZE08 * START/, V09A /SIZE09 * START/,
     .  V10A /SIZE10 * START/, V11A /SIZE11 * START/,
     .  V12A /SIZE12 * START/, V12B /SIZE12 * START/,
     .  V12C /SIZE12 * START/, V13A /SIZE13 * START/,
     .  V13B /SIZE13 * START/, V14A /SIZE14 * START/,
     .  V15A /SIZE15 * START/,
     .  V99A /SIZE99 * START/, V99B /SIZE99 * START/,
     .  V99C /SIZE99 * START/,
     .  M16A /SZ16SQ * START/, M16B /SZ16SQ * START/,
     .  M17A /SZ17SQ * START/
C
C  INITIALIZE THE STARTING MODE OF THE OPERATORS FOR THOSE LOOPS
C  THAT ALTERNATE BETWEEN ADD/SUBTRACT OR MULTIPLY/DIVIDE
C
      OP01 = ADD
      OP02 = ADD
      OP03 = MULT
      OP08 = ADD
      OP12 = ADD
      OP13 = MULT
      OP16 = ADD
      OP17 = MULT
C
C  INITIALIZE THE INTEGER ARRAYS TO 'RANDOM' VALUES THAT ARE
C  WITHIN THE PROPER RANGE.
C
      DO 1 I=1,SIZE15
        I15A(I) = I
1     CONTINUE
      DO 2 I=1,SIZE99
        I99A(I) = SIZE99-I+1
2     CONTINUE
      DO 3 I=1,SIZE99
        I99B(I) = I
3     CONTINUE
      DO 4 I=1,SIZE99
        I99C(I) = MAX(I99A(I),I99B(I))
4     CONTINUE
C-----------------------------------------------------------
C  BEGIN THE EXECUTION OF THE LOOPS
C-----------------------------------------------------------
      DO 1000 INDEX=1,LOOP
C
C  STATEMENTS IN WRONG ORDER
C
```

```
      IF (ABS(V01A(2)).GT.MAXNUM) OP01 = .NOT.OP01
      IF (OP01.EQV.ADD) THEN
        DO 10   I=2,N01
          V01B(I) = V01A(I-1)
          V01A(I) = V01A(I)+V99A(I)
10      CONTINUE
      ELSE
        DO 11   I=2,N01
          V01B(I) = V01A(I-1)
          V01A(I) = V01A(I)-V99A(I)
11      CONTINUE
      ENDIF
C
C  DEPENDENCY NEEDING TEMPORARY
C
      IF (ABS(V02B(2)).GT.MAXNUM) OP02 = .NOT.OP02
      IF (OP02.EQV.ADD) THEN
        DO 20   I=1,N02-1
          V02A(I) = V99A(I)
          V02B(I) = V02B(I)+V02A(I+1)
20      CONTINUE
      ELSE
        DO 21   I=1,N02-1
          V02A(I) = V99A(I)
          V02B(I) = V02B(I)-V02A(I+1)
21      CONTINUE
      ENDIF
C
C  LOOP WITH UNNECESSARY SCALAR STORE
C
      IF (ABS(V03A(2)).GT.MAXNUM .OR.
         ABS(V03A(2)).LT.MINFRC)      OP03 = .NOT.OP03
      IF (OP03.EQV.MULT) THEN
        DO 30   I=1,N03
          X = V99A(I)
          V03A(I) = V03A(I)*(X+V99B(I))
30      CONTINUE
      ELSE
        DO 31   I=1,N03
          X = V99A(I)
          V03A(I) = V03A(I)/(X+V99B(I))
31      CONTINUE
      ENDIF
C
C  LOOP WITH AMBIGUOUS SCALAR TEMPORARY
C
      T = 0.
        DO 40   I=1,N04
          S = V99A(I)*V99B(I)
          V04A(I) = S+T
          T = S
40      CONTINUE
C
C  LOOP WITH SUBSCRIPT THAT MAY SEEM AMBIGUOUS
C
        DO 50   I=1,N05/2
          V05A(I) = V05A(I+N05/2)
50      CONTINUE
C
```

```
C  RECURSIVE LOOP THAT REALLY ISN'T
C
      DO 60   I=1,N06-2,2
        V06A(I+1) = V06A(I)*4.
60    CONTINUE
C
C  LOOP WITH POSSIBLE AMBIGUITY BECAUSE OF SCALAR STORE
C
      DO 70   I=1,N07-1
        J = I+1
        V07A(I) = V07A(J)
70    CONTINUE
C
C  LOOP THAT IS PARTIALLY RECURSIVE
C
      IF (ABS(V08A(2)).GT.MAXNUM) OP08 = .NOT.OP08
      IF (OP08.EQV.ADD) THEN
        DO 80   I=2,N08
          V08A(I) = V08A(I)+(V99A(I)*V99B(I))
          V08B(I) = V08B(I-1)+V08A(I)+V99B(I)
80      CONTINUE
      ELSE
        DO 81   I=2,N08
          V08A(I) = V08A(I)-(V99A(I)*V99B(I))
          V08B(I) = V08B(I-1)+V08A(I)+V99B(I)
81      CONTINUE
      ENDIF
C
C  LOOP WITH UNNECESSARY ARRAY STORES
C
      DO 90   I=1,N09
        V09A(I) = V99A(I)+V99B(I)
        V09A(I) = V99C(I)*V09A(I)
90    CONTINUE
C
C  LOOP WITH INDEPENDENT CONDITIONAL
C
      T = 1.
      DO 100   I=1,N10
        IF (V99C(I).GE.T) THEN
          X = V99A(I)*V99B(I)+3.1
          Y = V99A(I)+V99B(I)*2.9
          V10A(I) = SQRT(X**2*Y)
        ENDIF
100   CONTINUE
C
C  LOOP WITH NONCONTIGUOUS ADDRESSING
C
      DO 110   I=1,N11,2
        V11A(I) = V99B(I)+3.*V99C(I)
110   CONTINUE
C
C  SIMPLE LOOP WITH DEPENDENT CONDITIONAL
C
      IF (ABS(V12A(2)).GT.MAXNUM) OP12 = .NOT.OP12
      IF (ABS(V12B(2)).GT.MAXNUM) OP12 = .NOT.OP12
      IF (ABS(V12C(2)).GT.MAXNUM) OP12 = .NOT.OP12
      IF (OP12.EQV.ADD) THEN
        DO 120   I=1,N12
```

```
          V12A(I) = V12A(I)+V99B(I)+V99C(I)
          IF (V12A(I).LT.0.) V12B(I) = V12B(I)+V99A(I)+V99B(I)
          V12C(I) = V12C(I)+V12A(I)+V99A(I)
120       CONTINUE
        ELSE
          DO 121    I=1,N12
            V12A(I) = V12A(I)-V99B(I)-V99C(I)
            IF (V12A(I).EQ.0.) V12B(I) = V12B(I)-V99A(I)-V99B(I)
            V12C(I) = V12C(I)-V12A(I)-V99A(I)
121       CONTINUE
        ENDIF
C
C  COMPLEX LOOP WITH DEPENDENT CONDITIONAL
C
        IF (ABS(V13B(2)).GT.MAXNUM .OR.
       .    ABS(V13B(2)).LT.MINFRC)      OP13 = .NOT.OP13
        IF (OP13.EQV.MULT) THEN
            DO 130    I=1,N13
              V13A(I) = V99A(I)+V99C(I)
              IF (V13A(I).EQ.0.) THEN
                 V13B(I) = V13A(I)*V13B(I)
              ELSE
                 V13A(I) = V99B(I)*V99C(I)
                 V13B(I) = 1.
              ENDIF
130         CONTINUE
        ELSE
            DO 131    I=1,N13
              V13A(I) = V99A(I)-V99C(I)
              IF (V13A(I).EQ.0.) THEN
                 V13B(I) = V13A(I)/V13B(I)
              ELSE
                 V13A(I) = V99B(I)/100.
                 V13B(I) = 1.
              ENDIF
131         CONTINUE
        ENDIF
C
C  LOOP WITH SINGULARITY HANDLING
C
            DO 140    I=1,N14
              IF (V99B(I).GT.0.) V14A(I) = V99B(I)/V99C(I)
140         CONTINUE
C
C  LOOP WITH SIMPLE GATHER/SCATTER SUBSCRIPTING
C
            DO 150    I=1,N15
              V15A(I15A(I)) =
                SQRT(V99A(I99A(I))*V99B(I99B(I))+V99C(I99C(I))**2+.5)
150         CONTINUE
C
C  LOOP WITH MULTIPLE DIMENSION RECURSION
C
        IF (ABS(M16A(2,2)).GT.MAXNUM) OP16 = .NOT.OP16
        IF (OP16.EQV.ADD) THEN
            DO 160    I=1,N16
            DO 160    J=2,N16
              M16A(I,J) = M16A(I,J-1)+M16B(I,J)
160         CONTINUE
```

```
      ELSE
          DO 161    I=1,N16
          DO 161    J=2,N16
            M16A(I,J) = M16A(I,J-1)-M16B(I,J)
161       CONTINUE
      ENDIF
C
C  LOOP WITH MULTIPLE DIMENSION AMBIGUOUS SUBSCRIPTS
C
        M = 1
        J = M
        K = M+1
      IF (ABS(M17A(2,2)).GT.MAXNUM .OR.
     .    ABS(M17A(2,2)).LT.MINFRC)     OP17 = .NOT.OP17
      IF (OP17.EQV.MULT) THEN
          DO 170    I=2,N17
            M17A(I,J) = M17A(I-1,K)*3.5
170       CONTINUE
      ELSE
          DO 171    I=2,N17
            M17A(I,J) = M17A(I-1,K)/3.5
171       CONTINUE
      ENDIF
1000  CONTINUE
      IF (PRTINC.NE.0) THEN
        WRITE(*,10001)
     .    (V01A(I),I=1,SIZE01,PRTINC), (V01B(I),I=1,SIZE01,PRTINC),
     .    (V02A(I),I=1,SIZE02,PRTINC), (V02B(I),I=1,SIZE02,PRTINC),
     .    (V03A(I),I=1,SIZE03,PRTINC),
     .    (V05A(I),I=1,SIZE05,PRTINC), (V06A(I),I=1,SIZE06,PRTINC),
     .    (V07A(I),I=1,SIZE07,PRTINC), (V08A(I),I=1,SIZE07,PRTINC),
     .    (V08B(I),I=1,SIZE07,PRTINC), (V09A(I),I=1,SIZE09,PRTINC),
     .    (V10A(I),I=1,SIZE10,PRTINC), (V11A(I),I=1,SIZE11,PRTINC),
     .    (V12A(I),I=1,SIZE12,PRTINC), (V12B(I),I=1,SIZE12,PRTINC),
     .    (V12C(I),I=1,SIZE12,PRTINC), (V13A(I),I=1,SIZE13,PRTINC),
     .    (V13B(I),I=1,SIZE13,PRTINC), (V14A(I),I=1,SIZE14,PRTINC),
     .    (V15A(I),I=1,SIZE15,PRTINC)
        WRITE(*,10002)
     .    ((M16A(I,J),I=1,SIZE16,PRTINC),J=1,SIZE16,PRTINC),
     .    ((M16B(I,J),I=1,SIZE16,PRTINC),J=1,SIZE16,PRTINC),
     .    ((M17A(I,J),I=1,SIZE17,PRTINC),J=1,SIZE17,PRTINC)
      ENDIF
C-------------------------------------------------------------
C FORMATS STATEMENTS FOLLOW
C-------------------------------------------------------------
10001 FORMAT('1VALUES OF SINGLE DIMENSION ARRAYS FOLLOW',/,(10E12.5))
10002 FORMAT('1VALUES OF DOUBLE DIMENSION ARRAYS FOLLOW',/,(10E12.5))
C
      STOP
      END
```

Appendix B: VECTOR Results

Summarized in Table B-1 are the results from the program VECTOR. As the table indicates, two CRAY compilers were tested: CFT 1.13 and CFT 1.15.

Table B-1
Loops Missed by the Respective Compilers

Loop Label	CRAY CFT 1.13	CRAY CFT 1.15	Fujitsu 77/VP v10l10	Hitachi fort77/hap (v02-00)
1				
2				
3				
4				
10, 11	X	X		
20, 21				
30, 31				
40	X	X	P	P
50			X	X
60				
70	X			
80, 81	X	X	P	
90				
100				
110				
120, 121				
130, 131				
140				
150				
160, 161				
170, 171			X	

X - Loop not vectorized.

P - Loop partially vectorized.

Below we present the information provided by each compiler about the nonvectorized loops.

CRAY X-MP CFT 1.15

.....

The following loops were not vectorized by the CRAY compiler.

```
C
C  STATEMENTS IN WRONG ORDER
C
      IF (ABS(V01A(2)).GT.MAXNUM) OP01 = .NOT.OP01
      IF (OP01.EQV.ADD) THEN
        DO 10   I=2,N01
          V01B(I) = V01A(I-1)
          V01A(I) = V01A(I)+V99A(I)
10      CONTINUE
      ELSE
        DO 11   I=2,N01
          V01B(I) = V01A(I-1)
          V01A(I) = V01A(I)-V99A(I)
11      CONTINUE
      ENDIF
```

Compiler message:

Dependency involving array V01A.

Previous minus with an incrementing subscript.

```
C
C  LOOP WITH AMBIGUOUS SCALAR TEMPORARY
C
    T = 0.
     DO 40  I=1,N04
      S = V99A(I)*V99B(I)
      V04A(I) = S+T
      T = S
40    CONTINUE
```

Compiler message.

No message given.

```
C
C  LOOP THAT IS PARTIALLY RECURSIVE
C
      IF (ABS(V08A(2)).GT.MAXNUM) OP08 = .NOT.OP08
      IF (OP08.EQV.ADD) THEN
        DO 80   I=2,N08
          V08A(I) = V08A(I)+(V99A(I)*V99B(I))
          V08B(I) = V08B(I-1)+V08A(I)+V99B(I)
80      CONTINUE
      ELSE
        DO 81   I=2,N08
          V08A(I) = V08A(I)-(V99A(I)*V99B(I))
          V08B(I) = V08B(I-1)+V08A(I)+V99B(I)
81      CONTINUE
      ENDIF
```

Compiler message.

Dependency involving array V08B.

Previous minus with an incrementing subscript.

(Note that partial vectorization takes place; there is no hardware to allow recursion as in the Hitachi S-810.

Fujitsu 77/VP v10l10

The following loops were not vectorized by the Fujitsu compiler.

```
C
C  LOOP WITH AMBIGUOUS SCALAR TEMPORARY
C
      T = 0.
        DO 40    I=1,N04
          S = V99A(I)*V99B(I)
          V04A(I) = S+T
          T = S
40      CONTINUE
```

Compiler message:
No message given for partial vectorization.

```
C
C  LOOP WITH SUBSCRIPT THAT MAY SEEM AMBIGUOUS
C
        DO 50    I=1,N05/2
          V05A(I) = V05A(I+N05/2)
50      CONTINUE
```

Compiler message:
Array V05A cannot be vectorized because recursive reference may take place.

```
C
C  LOOP THAT IS PARTIALLY RECURSIVE
C
      IF (ABS(V08A(2)).GT.MAXNUM) OP08 = .NOT.OP08
      IF (OP08.EQV.ADD) THEN
        DO 80    I=2,N08
          V08A(I) = V08A(I)+(V99A(I)*V99B(I))
          V08B(I) = V08B(I-1)+V08A(I)+V99B(I)
80      CONTINUE
      ELSE
        DO 81    I=2,N08
          V08A(I) = V08A(I)-(V99A(I)*V99B(I))
          V08B(I) = V08B(I-1)+V08A(I)+V99B(I)
81      CONTINUE
      ENDIF
```

Compiler message:
Some statements in this range cannot be vectorized since data dependency is recursive.

(Note that partial vectorization takes place; there is no hardware to allow recursion as in the Hitachi S-810.

```
C
C  LOOP WITH MULTIPLE DIMENSION AMBIGUOUS SUBSCRIPTS
C
        M = 1
        J = M
        K = M+1
      IF (ABS(M17A(2,2)).GT.MAXNUM .OR.
     .    ABS(M17A(2,2)).LT.MINFRC)     OP17 = .NOT.OP17
      IF (OP17.EQV.MULT) THEN
          DO 170    I=2,N17
            M17A(I,J) = M17A(I-1,K)*3.5
170       CONTINUE
```

```
      ELSE
           DO 171    I=2,N17
             M17A(I,J) = M17A(I-1,K)/3.5
171        CONTINUE
      ENDIF
```

Compiler message:
Variable K in subscript expression may cause recursive reference of array M17A
Variable J in subscript expression may cause recursive reference of array M17A
Relation between variables K and J may cause recursive reference.
Some statements in this range cannot be vectorized since the data dependency is recursive.

Hitachi fort77/hap (v02-00)

The following loops were not vectorized by the Hitachi compiler.

```
C
C  LOOP WITH AMBIGUOUS SCALAR TEMPORARY
C
      T = 0.
        DO 40    I=1,N04
          S = V99A(I)*V99B(I)
          V04A(I) = S+T
          T = S
40      CONTINUE
```

Compiler message
DO LOOP 40 is partially vectorizable
V04A(I) = S+T, variable T used before definition

```
C
C  LOOP WITH SUBSCRIPT THAT MAY SEEM AMBIGUOUS
C
        DO 50    I=1,N05/2
          V05A(I) = V05A(I+N05/2)
50      CONTINUE
```

Compiler message
Unknown list vector data dependency in variable V05A.

Distribution for ANL-85-19

Internal:

J. J. Dongarra (40)
A. Hinds (40)
K. L. Kliewer
A. B. Krisciunas
P. C. Messina
G. W. Pieper
D. M. Pool
T. M. Woods (2)

ANL Patent Department
ANL Contract File
ANL Libraries
TIS Files (6)

External:

DOE-TIC, for distribution per UC-32 (167)
Manager, Chicago Operations Office, DOE
Mathematics and Computer Science Division Review Committee:
- J. L. Bona, U. Chicago
- T. L. Brown, U. of Illinois, Urbana
- S. Gerhart, MCC, Austin, Texas
- G. Golub, Stanford University
- W. C. Lynch, Xerox Corp., Palo Alto
- J. A. Nohel, U. of Wisconsin, Madison
- M. F. Wheeler, Rice U.

D. Austin, ER-DOE
J. Greenberg, ER-DOE
G. Michael, LLL